torch.nn 与torch.nn.functional的区别于联系

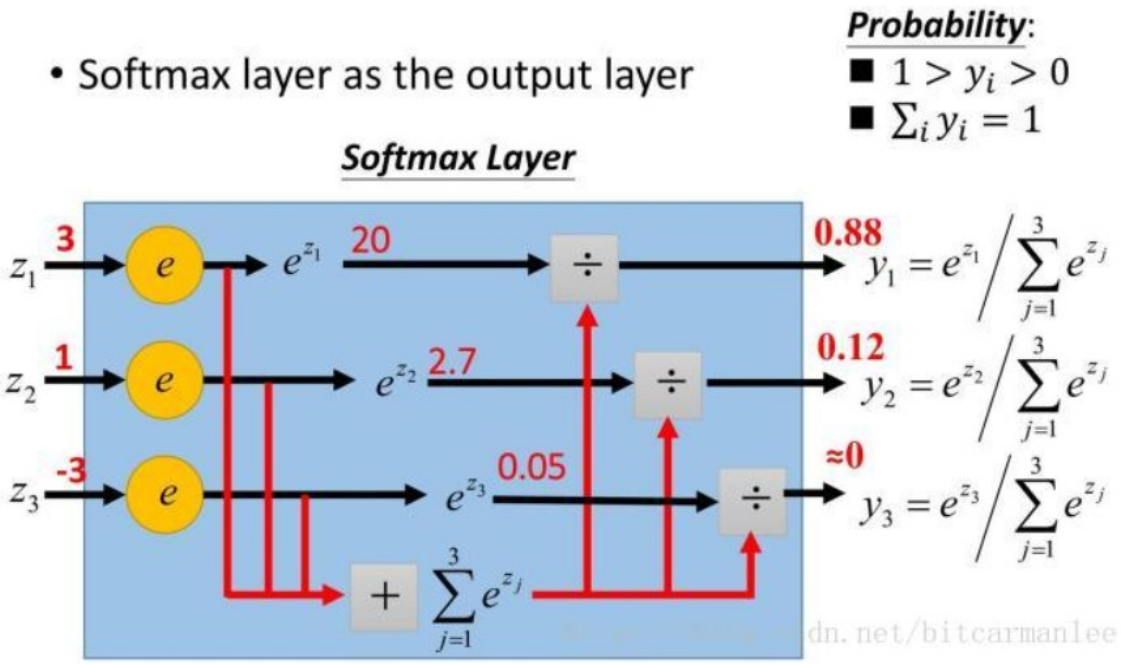
```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

区别：torch.nn中的模块可以保存参数的信息，而functional模块中需要在每次调用时传参；

联系：torch.nn模块中的功能都是调用functional模块来实现，也就是说两者功能保持一致，只是torch.nn是在functional的基础上在外面包了一层，用于存储参数信息。

1. softmax

softmax的数学含义如下图所示：



假设有一个数组V，V_i表示V中的第i个元素，那么这个元素的softmax值为:

$$S_i = \frac{e^i}{\sum_j e^j}$$

简言之，softmax的操作可以放大数组元素之间的差异，且同时保持所有经过softmax之后的数组，所有元素值的和为1，因为可以通过softmax的值来代表概率，用于分类网络中计算交叉熵损失。

nn.Softmax()

对n维输入张量运用Softmax函数，将张量的每个元素缩放到（0,1）区间且和为1。Softmax函数定义如下：

$$f_i(x) = \frac{e^{(x_i - shift)}}{\sum^j e^{(x_j - shift)}}, shift = max(x_i)$$

shape：

- 输入：(N, L)
- 输出：(N, L)

返回结果是一个与输入维度相同的张量，每个元素的取值范围在（0,1）区间。

使用示例：

```
m = nn.Softmax(dim=2)         # 可以先定义出接口，其中dim参数可选，默认是1
input = torch.rand((3,4,5,6))
output = m(input)
print(torch.sum(output, dim=2))     # 此时输出全为1
```

F.softmax()

与nn.Softmax的功能一样，使用方法如下：

```
input = torch.rand((3,4,5,6))
output = F.softmax(input, dim=2) # 直接调用，且设置dim，默认为1
print(torch.sum(output, dim=2))  # 此时输出全为1
```

2. Log softmax

log softmax的数学含义是：在softmax的基础上再进行一次log操作，如下所示：

log_softmax(input) = log(softmax(input))

因为softmax之后的概率值都是0~1范围内，所以经过log操作后，所有的数值都变为负值。

nn.LogSoftmax()

使用示例：

```
m = nn.LogSoftmax(dim=2)         # 可以先定义出接口，其中dim参数可选，默认是1
input = torch.rand((3,4,5,6))
output = m(input)
```

F.log_softmax()

使用示例：

```
input = torch.rand((3,4,5,6))
output = F.log_softmax(input, dim=2) # 直接调用，且设置dim，默认为1
# 验证
output2 = torch.log(F.softmax(input, dim=2))
print(output.equal(output2))     # 结果为True，说明两者结果一致
```

3. NLL loss

nll loss是指 log likelihood loss，即负对数似然损失。

输入是包含类别log probabilities的数据，因此一般需要在网络的最后一层增加一个求log的操作层，常见的是使用log_softmax

损失的计算是可以看如下描述：

此 `loss` 期望的 `target` 是类别的索引 (0 to N-1, where N = number of classes)

此 `loss` 可以被表示如下：

$$loss(x, class) = -x[class]$$

如果 `weights` 参数被指定的话， `loss` 可以表示如下：

$$loss(x, class) = -weights[class] * x[class]$$

nn.NLLLoss()

参数说明：

- weight (Tensor, optional) – 手动指定每个类别的权重。如果给定的话，必须是长度为 `nclasses`
- size_average (bool, optional) – 默认情况下，会计算 `mini-batch``loss` 的平均值。然而，如果 `size_average=False` 那么将会把 `mini-batch` 中所有样本的 `loss` 累加起来。

形状：

- Input: (N,C) , `C` 是类别的个数
- Target: (N) , `target` 中每个值的大小满足 `0 <= targets[i] <= C-1`

使用示例：

```
m = nn.LogSoftmax()      # 定义logsoftmax
loss = nn.NLLLoss()      # 定义nll loss
# input is of size nBatch x nClasses = 3 x 5
input = autograd.Variable(torch.randn(3, 5), requires_grad=True) # 创建input
# each element in target has to have 0 <= value < nclasses
target = autograd.Variable(torch.LongTensor([1, 0, 4]))  # 标签
output = loss(m(input), target)      # 计算loss
output.backward()   # 反向传播
```

F.nll_loss()

使用示例：

```
# input is of size nBatch x nClasses = 3 x 5
input = autograd.Variable(torch.randn(3, 5), requires_grad=True) # 创建input
# each element in target has to have 0 <= value < nclasses
target = autograd.Variable(torch.LongTensor([1, 0, 4]))  # 标签
output = F.nll_loss(F.log_softmax(input), target)      # 计算loss
output.backward()   # 反向传播
```

4. cross entropy loss

交叉熵损失用于分类网络中，在pytorch中的交叉熵损失相当于logsoftmax与nllloss的结合。

此标准将 `LogSoftMax` 和 `NLLLoss` 集成到一个类中。

当训练一个多类分类器的时候，这个方法是十分有用的。

- weight(tensor): `1-D` tensor，`n` 个元素，分别代表 `n` 类的权重，如果你的训练样本很不均衡的话，是非常有用的。默认值为None。

调用时参数：

- input : 包含每个类的得分，`2-D` tensor, `shape` 为 `batch*n`
- target: 大小为 `n` 的 `1-D` `tensor`，包含类别的索引(`0到 n-1`)。

Loss可以表述为以下形式：

$$loss(x, class) = -\log\frac{exp(x[class])}{\sum_j exp(x[j]))} = -x[class] + log(\sum_j exp(x[j]))$$

当 `weight` 参数被指定的时候，`loss` 的计算公式变为：

$$loss(x, class) = weights[class] * (-x[class] + log(\sum_j exp(x[j])))$$

计算出的 `loss` 对 `mini-batch` 的大小取了平均。

形状(`shape`)：

- Input: (N,C) `C` 是类别的数量
- Target: (N) `N` 是 `mini-batch` 的大小，0 <= targets[i] <= C-1

nn.CrossEntropyLoss()

输入参数的设置与Nll loss的设置一样。

使用示例:

```
# input is of size N x C = 3 x 5
input = torch.randn(3, 5, requires_grad=True)     # 输入input
# each element in target has to have 0 <= value < C
target = torch.tensor([1, 0, 4])          # 标签
loss = nn.CrossEntropyLoss(dim=1) # 定义ce loss
output = loss(input, target) # 计算 loss
print(output)
```

F.cross_entropy()

使用示例：

```
input = torch.randn(3, 5, requires_grad=True)         # 输入input
# each element in target has to have 0 <= value < C
target = torch.tensor([1, 0, 4])
loss = F.cross_entropy(input, target)
loss.backward()
```

5. BCE loss

计算 `target` 与 `output` 之间的二进制交叉熵

当不指定weights时：

$$loss(o,t) = -\frac{1}{n}\sum_i (t[i]log(o[i]) + (1-t[i])log(1-o[i]))$$

当指定weights时：

$$loss(o,t) = -\frac{1}{n}\sum_i weights[i](t[i]log(o[i]) + (1-t[i]) * log(1-o[i]))$$

默认情况下，loss会基于`element`平均，如果`size_average=False`的话，`loss`会被累加。

nn.BCELoss()

使用示例：

```
m = nn.Sigmoid()
loss = nn.BCELoss()
input = torch.randn(3, requires_grad=True)
target = torch.empty(3).random_(2)
output = loss(m(input), target)
output.backward()
```

F.bce_loss()

使用示例：

```
input = torch.randn((3, 2), requires_grad=True)
target = torch.rand((3, 2), requires_grad=False)
loss = F.binary_cross_entropy(F.sigmoid(input), target)
loss.backward()
```

参考链接

1. https://blog.csdn.net/qq_22210253/article/details/85229988
2. https://www.jianshu.com/p/35060b7553c8
3. https://blog.csdn.net/geter_CS/article/details/84857220
4. https://blog.csdn.net/hao5335156/article/details/80607732
5. https://www.cnblogs.com/wanghui-garcia/p/10862733.html
6. https://blog.csdn.net/shanglianlm/article/details/85019768